Improvements Report on Project “Milly”

Current Topics of Computer Science
by Matthias Pfuhl

What has changed?

To jump much deeper into AI development I decided to implement a min-max search algorithm with variable search depth and customizable board evaluation.
Especially the evaluation changes the outcome of the game and the playstyle of the AI a lot.
I used the work of [GASS] to improve major decisions an AI should make when it comes to the placing phase, choosing a move, jumping and removing a piece of an enemy.

Min-Max Algorithm

My Nine Men’s Morris AI’s so far were only evaluation the current board situation.
But every Nine Men’s Morris player is thinking several turns ahead and you can’t decide which move is the best if you are not evaluate all possible moves now and (some) of the future moves. A basic approach for game AI developers, when it comes to two-player games, is the min-max algorithm.

In theory - with min-max the AI is simulating all moves you can do now, then all moves your opponent can do, then all your possible moves again (of course in dependency of the opponent’s move before) and so on, until the game ends. Then the algorithm calculates a board-state-value for the outcome (e.g. 1 for a win and 0 for a loss) and with going recursively backwards one move that led to a win is chosen as the next move.

Practically, for Nine Men’s Morris, that is of course not possible in a satisfying time window. So what one can do is choosing a search depth so that the algorithm is going down the turns tree until that depth is reached and then again calculating the current board state. With going recursively backwards, if that turn was your own you want to maximize the board-state-values of that tree level and copy the max value to the parent. If you are on a tree level of your opponent’s possible moves, you want to minimize that value and copy it to the parent.
If you end up on the top again (or the tree root if you want) you simply chose the move with the highest value, which means: “If you do that, you’ll end up in a good position”.

Basic Min-Max and Evaluation

For the basic min-max I used following pseudo code:

MAINPROGRAM:

```
savedMove = NULL;
int searchDepth = 4;
int value = max(+1, searchDepth);
if (savedMove == NULL)
    “there are no more moves possible”;
else
    do savedMove;
```

MAX:

```
int max(int player, int depth) {
    if (depth == 0 or noMovesPossible(player))
       return evaluate();
    int maxValue = -infinity;
    getAllPossibleMoves(player);
    while (one more move exists) {
       doMove(move);
       int value = min(-player, depth-1);
       undoMove(move);
       if (value > maxValue) {
          maxValue = value;
          if (depth == searchDepth)
             savedMove = move;
       }
    }
    return maxValue;
}
```

MIN:

```
int min(int player, int depth) {
    if (depth == 0 or noMovesPossible(player))
       return evaluate();
    int minValue = infinity;
    getAllPossibleMoves(player);
    while (one more move exists) {
       doMove(move);
       int value = max(-player, depth-1);
       undoMove(move);
       if (value < minValue) {
          minValue = value;
          if (depth == searchDepth)
             savedMove = move;
       }
    }
    return minValue;
}
```

For the basic evaluation I used the work of Gasser [GASS] and a Nine Men's Morris strategy website [SBD] and the evaluation supposed to use following values:

```
cornerPosVal = 2;
smallIntersecPosVal = 3;
bigIntersecPosVal = 5;
millVal = 4;
unmoveableManVal = -5;
```

In the `evaluate()` method, those values are added to a sum of all of your and subtracted from that sum for all of your enemies men. So that for example the value of the current board state (and therefore for the move you did) is positive if you could form a mill.
The idea is that of course one move is maybe not that good, but if that move leads to a mill in your next turn, you should do that move. That is the intention of min-maxing.
As I said the evaluation values are based more or less on some strategy advices from [SBD]. For example a "corner position" (`cornerPosVal`) is not very preferable but if you move like you'll end up with an "unmoveable man", that move would be pretty bad and that's why I chose a negative value here.
The value for a mill is in that example 12, since there are 3 pieces to form that mill.

Improvements on Min-Max

After the first couple of games of the basic min-max AI versus itself or my aggressive rule-based AI, I recognized a few things:

- **The search depth needs to be customizeable.** Search depth of 4 was a good start but if you think of it, you can see yourself and the opponent only doing 2 moves and especially if your pieces are very separated you can't see the best move yet.
- **The search needs to make trivial decisions.** Before even starting to min-max all possible moves, if there is a trivial move like "filling a mill","finishing the game" or "block your opponents mill" (in placing phase)
- **The jumping phase doesn't need to min-max.** In the jumping phase it's all about finishing the game. So the AI just needs to open and close the mill, block the opponent's mill and be aware of not open the mill dangerously (for example if there is at least one opponent's piece able to move to that position - it's dangerous since you can't remove that piece now). It's all about very trivial or strict decisions to make in that phase, that's why this AI doesn't need to min-max here.

Improvements on Evaluation

Doing trivial moves before min-maxing was a big upgrade for my AI, but what really bothered me were the evaluation values.
I thought: Maybe if one changes the values in one or the other direction could lead to a more aggressive or more defensive game AI.
For example corners are really bad, but how bad?
Or big intersections are really really nice to have in every game state but are they really only slightly better than small intersections?

And should you maybe also evaluate the kind of neighbours you have at a intersection.
I figured out that the evaluation of the current board state itself could be an own project, because what's really good can only be said after lots and lots of tests.

I changed some values, for example did I lower the values for the opponent so that my AI would act more aggressive and is going to prefer moves that lead to an own mill instead of moves that are going to block enemy mills.
But is that really better?
I can't say that and even after my tests (1000 games, normal AI wins: 561, aggressive AI wins: 439) I'm not sure if that is really a good direction for a Nine Men's Morris AI.
Maybe it would be good to interview some professional players on how they would evaluate some states and one could implement an AI after his/her thoughts.
But of course even they are doing mistakes and obviously no evaluation would be the best.

Another approach would be to "test" some evaluations and change them slightly from game to game and then change it back if the AI is keep losing that way or stay with the change if it is winning more.

References

[GASS] Gasser Ralph, 1996. Solving Nine Men's Morris. [pdf] Available at: http://library.msri.org/books/Book29/files/gasser.pdf [Accessed December 2016].

[SBD] Smart Box Design, 2010. *Triples Strategies*. [online]
Available at: <http://www.smartboxdesign.com/ninemmstrategies.html>
[Accessed 27 January 2017].